Effects of the Number of Developers on Code Quality in Open Source Software: A Case Study

Brandon Norick, Justin Krohn, Eben Howard, Ben Welna, Clemente Izurieta

Department of Computer Science

Montana State University

Bozeman, MT 59717

01-406-994-4780

{brandon.norick, justin.krohn, eben.howard, ben.welna}@msu.montana.edu, {clemente.izurieta}@cs.montana.edu

ABSTRACT

Eleven open source software projects written in C/C++ were analyzed to determine if the number of committing developers impacts code quality. We use cyclomatic complexity, lines of code per function, comment density, and maximum nesting as surrogate measures of code quality. We find no significant evidence to suggest that the number of committing developers affects the quality of software.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *software evolution, product metrics, code quality.*

General Terms

Measurement, Design, Experimentation.

Keywords

Code quality, open source software, number of developers.

1. INTRODUCTION

Several research projects have investigated code quality in open source projects [1-3, 11]. In general, findings indicate that open source code suffers from poor quality. The motivation for this exploratory case study is to investigate if quality is affected by the number of committing developers involved in the project. We chose eleven mature open source projects; Filezilla, WinSCP, Miktex, UltraVNC, Notepad++, Audacity, Exult, SMPlayer, TCL, Wireshark, and TortoiseSVN. Selected projects were deemed mature by observing the total number of downloads. We used SourceForge's "Top Downloads All Time Metric."

Contrary to prior results, we find evidence to indicate that all eleven open source projects show characteristics of good code quality; the exception being TCL, which displays trends that are incompatible when compared to the other projects in the case study. Further, we find that there appears to be no significant evidence to indicate a relationship between the number of developers involved in a project and the observed code quality metrics.

2. RELATED WORKS

Although some research [5] finds that open source software shows signs of increased creativity, higher complexity, and faster bug fixes in open source software when compared to similar closed source projects, there exists counter examples to suggest otherwise [11]. Stamelos et. al. find an average comment density of 10% for 100 Linux applications studied. This finding persuaded the authors to posit the possibility that open source software may sacrifice quality and good coding practices to focus on bug fixes and new features instead. The metrics gathered for the Linux applications in [11] served as a baseline and reference for comparing our case study results against an established and successful domain. Stamelos et. al. found that the average number of statements per function was approximately 24, the average cyclomatic complexity was 7.7, the maximum nesting was approximately 3, and the comment density was approximately 11%.

Previous work indicates that organizational metrics are able to more correctly detect error-prone code than the usage of more traditional quality surrogates such as cyclomatic complexity or churn [4]. These metrics refer to the human organizational aspects of developer groups creating the software, rather than the software itself. Also, additional research suggests that organizational metrics provide the best indicator of fault-proneness in software [1-3, 6]. Unfortunately, organizational information is not immediately available from open source repositories and thus cannot be used in case studies. We believe that while this metric may not be obtainable, a general sense of the organizational structure would help interpret other metrics.

3. CASE STUDY DESIGN

3.1 Metrics

The following subsections describe the set of metrics that were used as surrogates for code quality.

3.1.1 McCabe's Cyclomatic Complexity

Despite reservations from some researchers, cyclomatic complexity is often viewed as a good indicator of the effort required to produce and maintain code [7, 9]. In this study, cyclomatic complexity indicates good code quality if procedural code scores under 10 in complexity. Object oriented software has a lower score due to modularized complexity being shared across multiple methods [7, 11].

3.1.2 *Lines of Code*

Research suggests that approximately fifty lines per function is considered ideal [7]. There also exists research that demonstrates a correlation between lines of code and cyclomatic complexity [9, 11]. A discrepancy between these metrics should be analyzed carefully.

3.1.3 *Comment Density*

Research indicates that the percentage of comments in code should be about 30% [10-11]. This metric suggests code is well documented, and therefore relatively easy to maintain and debug. Higher percentages are indicative of overly complex code.

3.1.4 *Maximum Nesting*

Maximum nesting (measured over the total lines of source code) is an indicator of the maximum complexity of the code [11]. As this number increases, the code becomes more difficult to understand and the effort required to debug and modify the code also increases.

3.2 Methodology

3.2.1 *Selecting a Discovery Tool*

Various tools were evaluated. We selected *Understand* from SciTools [14]. Amongst its functionality, this application is able to parse source code of any size. The application also provides support for a wide variety of metrics, including this study's surrogate selection of quality metrics. To ensure that the same metrics were selected for each project, we restricted our analysis to only C/C++. Additionally, we focus only on the OSS code, thus no external libraries were included in the analysis.

3.2.2 *Selecting the Projects*

In order to determine if the number of developers had an impact on code quality, we chose a set of projects where the number of developers varied significantly. Only projects written in C/C++ were chosen. Many projects were assessed, and this part of the process proved challenging due to the lack of consistency in stored information from one project to another. In some cases, we were unable to access the source code repository for a project, which made determining the number of developers impossible. We turned to SourceForge's "Most Active All Time" project rankings to select projects that gave us a broad range of open source software projects.

Each projects' latest version was downloaded from its respective website and a corresponding project was created in *Understand*. Some versions were marked beta rather than official releases. The beta versions were included because some open source projects had an official release version that was older than two years old, yet the vast majority of downloads were for the latest beta version.

3.2.3 *Determining the Number of Developers*

Before determining the number of developers participating in a project, we need a characterization. This is a difficult task in OSS because precise definitions vary significantly. Many questions arise; for example, does someone who only changes few lines count as a developer, or does a *bug fixer* count as a developer. We decided to characterize a developer as an individual with change and source control responsibilities. In the case of OSS, this meant that only developers committing to the source repository would be counted. A committing developer is ultimately responsible for change, and is therefore a good measure of how many people 'developed' a given software entity. A further refinement we took was to only look at committing developers that fell within the ninety day window prior to the release version we were investigating.

To determine the number of developers working on a project we process the log files generated by different source repositories. Each source repository we mined allowed us to export its respective log files. For each project we generated logs for the ninety day period that preceded the release of the version we studied. We used StatCVS [12] and StatSVN [13] to analyze the log files. The tools produced developer contribution statistics which we were able to use to determine the number of developers. Additionally, we manually analyzed specific developer data, and ruled out any "developers" which fell outside our characterization. The manual process was necessary to rule out situations where an automated process would perform a commit in the logs using its own account, or the cases where developers switch accounts but are registered more than once.

3.2.4 *Evaluating Code Quality*

All selected projects were written in C/C++ with some projects containing additional code in other languages. Only the C/C++ sections of the code were analyzed. No external libraries were analyzed. We used *Understand* on every project to obtain the cyclomatic complexity, lines of code, comment density, and maximum nesting metrics. The cyclomatic complexity metric was calculated as the average value of all methods in the source code. The maximum cyclomatic complexity was also reported, but was not used for comparison purposes. The maximum nesting metric was not used as a primary indicator of code quality, but was instead used in some situations where other metrics were ambiguous due to unusual code structuring.

4. RESULTS

Results suggest that the number of committing developers is not an indicator of code quality. Ten out of eleven projects showed signs of good code quality regardless of the number of developers involved. One project (i.e., TCL) was sufficiently different in code style and purpose, that when compared with the other projects, its apparently poor quality metrics can be attributed to its intended use.

4.1 Results Data

The following figures are sorted in ascending order from the least number of committing developers to the greatest. Figure 1 displays the average cyclomatic complexity of each project. In general, a value greater than 10 is considered to represent poor code quality in procedural code [7]. For object oriented software an exact value for this metric has not yet been proposed, but conjecture suggests that a lower number is expected. Cyclomatic complexity results show that all projects fall within the standard definition of good quality. Even TCL, with its complexity measured at just over 10, is within the range of good quality source code.

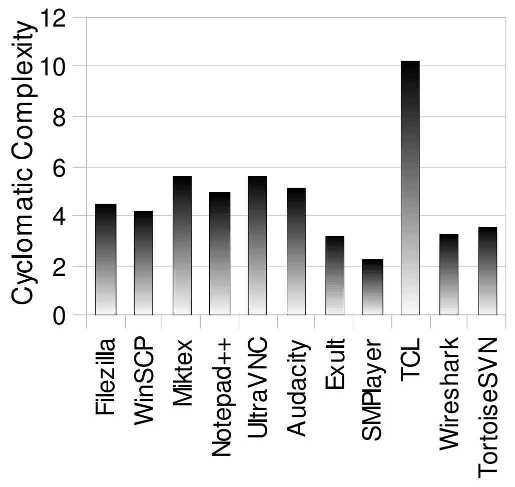

Figure 1. Average Cyclomatic Complexity

The average number of lines of code per function is shown in Figure 2. Approximately fifty lines per function are generally considered to be a good value for this metric. All projects, with the exception TCL, appear to have acceptable deltas from the suggested metric. Because TCL is made up of a very large number of comments this figure is slightly misleading.

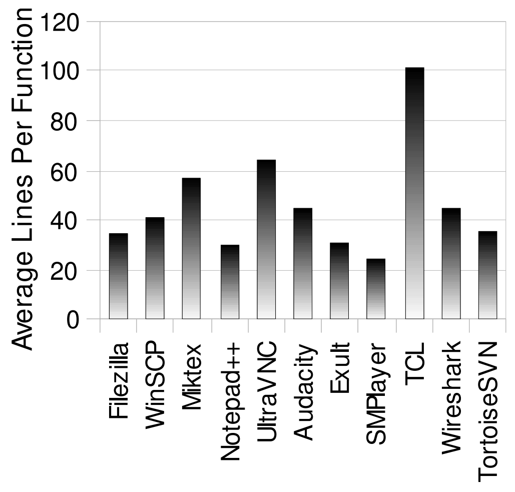

Figure 2. Average Number of Lines of Code per Function

Figure 3 displays the comment density for every project. A value in the range of 30% with a small standard deviation represents well documented code [3] and an agreed upon balance between too little and too much documentation. TCL is again the outlier value; however this is expected of a system designed with extensibility in mind. As a framework, it is expected that the average number of lines of code and comment density values measured per function will be significantly larger.

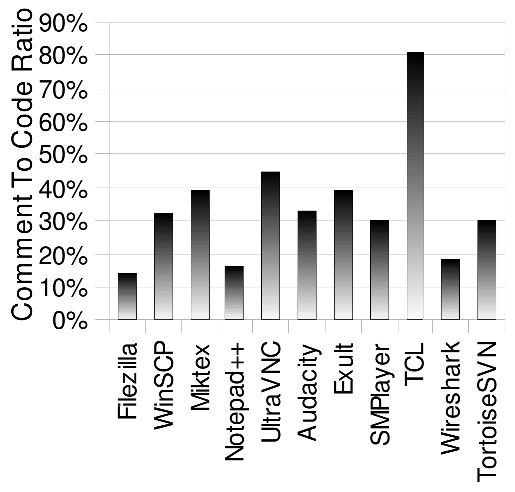

Figure 3. Average Comment Density

Figure 4 displays the maximum nesting in the source code. This is generally regarded as a measure of maximum complexity, with a lower complexity considered a sign of better code quality. Generally, this metric has a positive correlation with other quality metrics, and is indicative of poor quality code only when other metrics also indicate poor quality [11]. All projects, including TCL fall within the accepted range of good quality code.

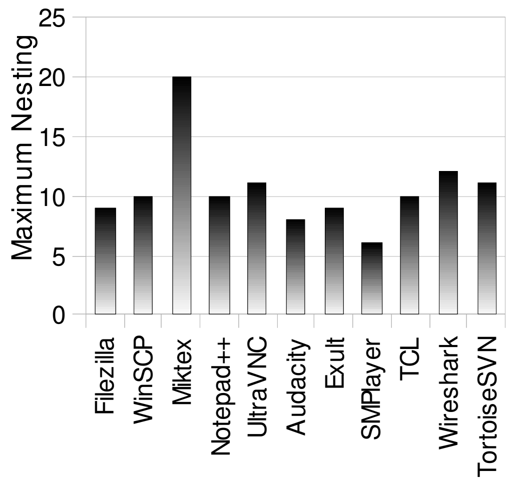

Figure 4. Maximum Nesting of Code per Function

Table 1 shows Pearson and Spearman correlation coefficient values generated via simple regression analysis. Cyclomatic complexity exhibits higher values; however any evidence of being correlated with the total number of committing developers is not significant. Additionally, all values obtained for cyclomatic complexity in all projects were within agreed upon bounds of good code quality. Even though we do not have enough data to make use of parametric tests, we chose to include them because they serve as a reference to Spearman's results.

Table 1. Correlation coefficients between quality metrics and the total number of committing developers

Quality Metric	Pearson's r^2	Spearman's r^2
Comment to code ratio	0.000225	0.003249
Average lines per function	0.003844	0.000081
Maximum nesting	0.001296	0.001681
Cyclomatic complexity	0.022801	0.084681

4.1.1 Discussion

All projects examined, with the exception of TCL, showed signs of good code quality when compared to suggested quality metrics. TCL is a developer's framework written entirely in procedural style; and thus some metrics do not correlate with other projects in terms of quality. However, the cause is likely to be TCL's code style and intent rather than poor design. As a framework, TCL requires a large amount of commenting in order to be used and extended by developers not familiar with the code.

The comment density of some projects showed a percentage almost as low as 10%, which has been noted by other research in open source software [3]. Most of the projects show about the right percentage of comments to code, with TCL being the obvious outlier.

4.2 Evaluation of Validity

Consistent with Runeson and Host [8] guidelines for conducting empirical research in software engineering, we evaluate the threats to the validity of this study.

The metrics selected as surrogate measures for quality represent a small operational subset that could be expanded to include additional measures. Coupling measures will extend the representation of the meaning of quality and will be added in the future to improve the construct validity. Further, in order to improve external validity, we clearly need to expand on the number of systems; however it should be noted that statistically significant samples sizes in case studies are not, in general, possible. We have no evidence to suggest that there exists a reliability threat to validity and any competent researcher can reproduce the results under similar study conditions. Finally, internal validity refers to the examination of possible causal relationships that may exist in the data. Whilst we investigated whether the number of committing developers affects code quality, we found no significant evidence to suggest the latter. The risk of third factors affecting code quality is therefore not an issue in this study.

5. CONCLUSION & FUTURE WORKS

After mining eleven open source projects we have no significant evidence to suggest that there is a link between code quality and the number of committing developers. Contrary to past studies that suggest that too many or too few developers will decrease the quality of software, we find no such evidence [1].

In order to improve external validity of our results we intend to increase the number of projects as well as the number and quality of metrics used. Current work to determine statistical significance of the results is underway. This will help us determine expected averages and analyze standard deviations from expected norms. Additionally, a comparison to similar closed source projects could be enlightening. An evaluation of closed source software similar in function to TCL and MikTex would further help us understand differences in code quality as the relationship to the number of developers.

6. REFERENCES

[1] de Groot, A., Kügler, S., Adams, P.J., and Gousios G. "Call for Quality: Open Source Software Quality Observation," *IFIP International Federation for Information Processing*, vol. 203/2006, pp. 57-62, August 2006.

[2] Mockus, A., Fielding, R. T., and Herbsleb, J. "A Case Study of Open Source Software Development: The Apache Server," *Proceedings of the 22nd international conference on Software Engineering, pp.263-272,* 2000

[3] Mockus, A., Fielding, R. T., and Herbsleb, J. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol 11/2002, issue 3, pp. 309-346, 2002

[4] Nagappan, N., Murphy, B., and Basili, V. R. "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," *ICSE '08,* pp. 521-530, 2008

[5] Paulson, J. W., Succi, G., and Eberlein, A. "An Empirical Study of Open-Source and Closed-Source Software Products.,"*IEEE Transaction on Software Engineering*, vol 30, no. 4, pp. 246-256, 2004

[6] Pendharkar, P.C. and Rodger, J. A. "An Empirical Study of the Impact of Team Size on Software Development Effort," *Information Technology Management*, vol 8, issue 4, pp. 253-262, 2007

[7] Prasad, L. and Nagar, A. "Expreimental Analysis of Different Metrics (Object-oriented and Structual) of Software," *2009 First International Conference on Computational Intelligence, Communication Systems and Networks*, pp. 235-240, 2009

[8] Runeson, P. and Host, M. "Guidelines for conducting and reporting case study research in software engineering," Journal of Empirical Software Engineering, Vol 14 pp. 131-164, 2009

[9] Shepperd, M. "A Critique of Cyclomatic Complexity as a Software Metric," *Software Engineering Journal*, vol 3, issue 2, pp. 30-36,3/1988

[10] Simmons, M. M., Vercellone-Smith, P., and Laplante, P.A. "Understanding Open Source Software through Software Archaeology: The Case of Nethack," *30th Annual IEEE/NASA Software Engineering Workshop SEW-30*.pp. 47-58, 2006

[11] Stamelos, I., Angelis, L., Oikonomou, A., and Bleris, G. L. "Code Quality Analysis in Open Source Software Development," *Info Systems Journal*, vol 12, pp. 43-61, 2002

[12] "StatCVS 0.7.0", http://statcvs.sourceforge.net/

[13] "StatSVN 0.7.0", http://www.statsvn.org/

[14] "Understand", *SciTools*, http://www.scitools.com/